Real-time Operating Systems and Systems Programming

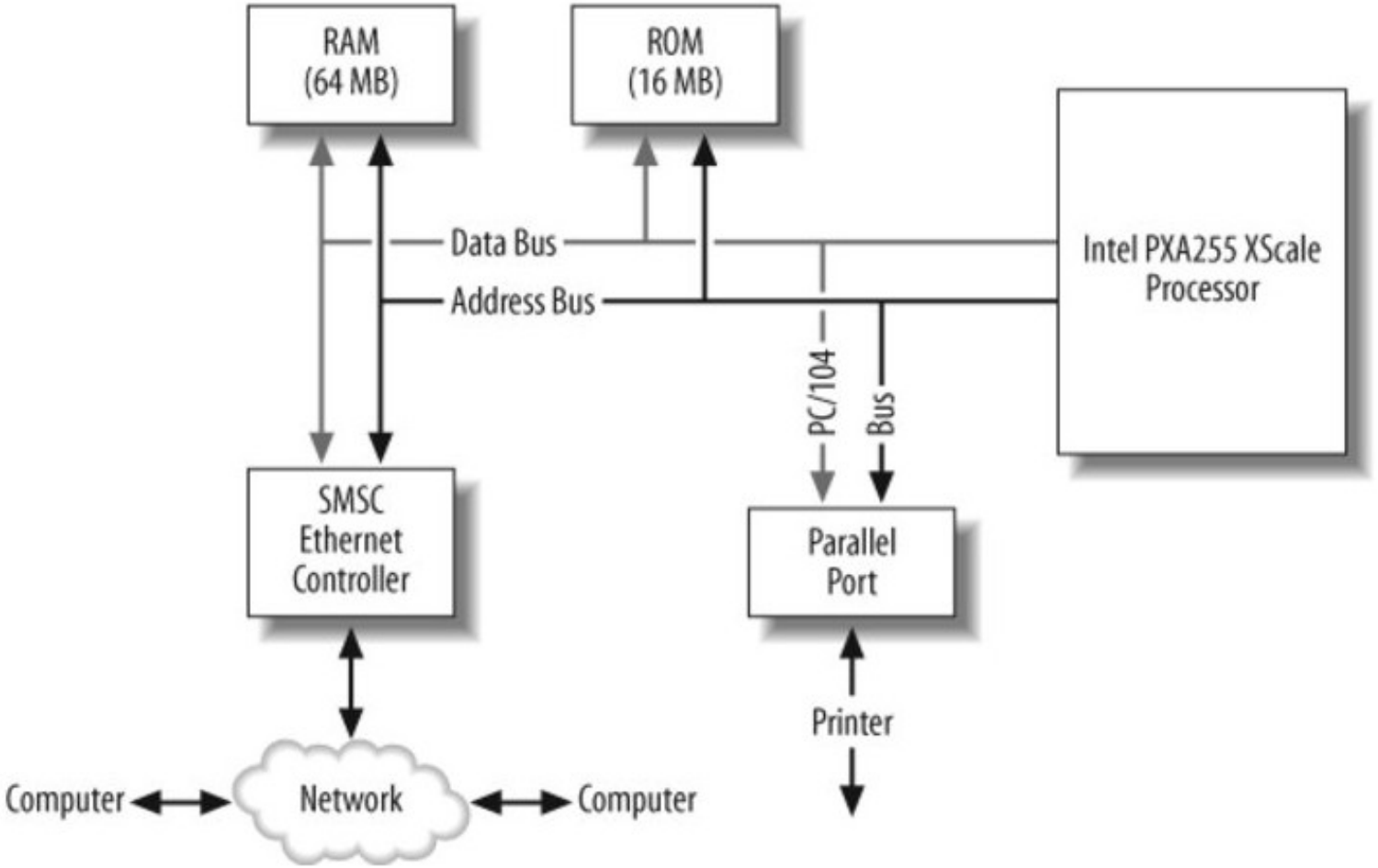
Getting to Know the Hardware
Lecture 12

ARVUTITEADUSE INSTITUUT
DEPARTMENT OF COMPUTER SCIENCE

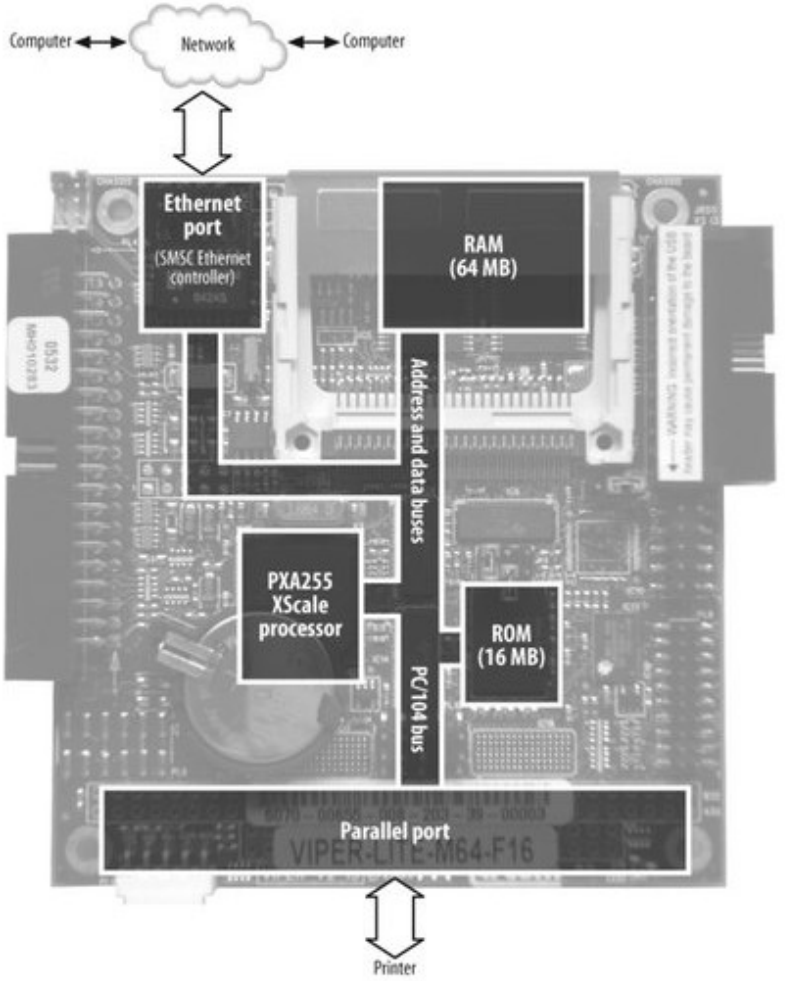
Big Picture

- Get to know the general operation of the system
- Read the main documents for programmers
  - User's Guide; Programmer's Manual
- What is the overall purpose?
- How does the data flow through it?
- If confused, get a hardware expert to explain

Example diagram



Overlay diagram



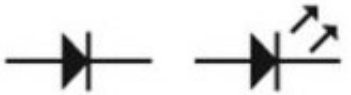
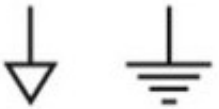
Aside: Project binder

- Useful for reference
- Previous diagram would be there
- Add all the documentation
  - Memos
  - Design justifications
- Use tabs to organize
- Will be absolutely amazing when you have to revise the project months years later

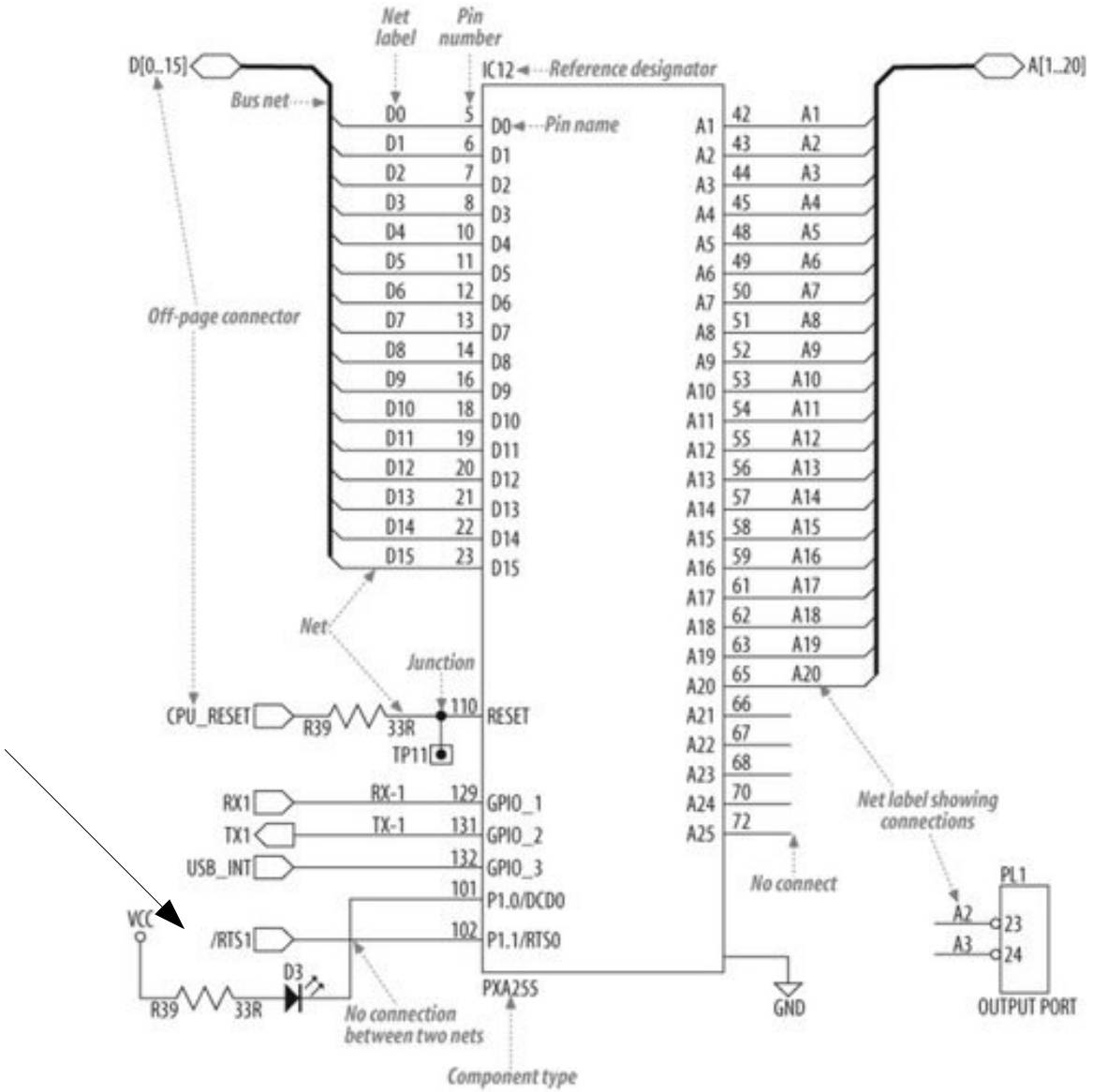
Understanding Schematics

- Data schematics provide a standardized way to representing the hardware, its connections, components
- Collect datasheets for main components
  - Timing and interface parameters, occasional code
- Errata documents document issues and their workaraounds

Fundamental Schematics

Component	Reference Designator Prefix	Symbol
Resistor	R	
Capacitor	C	
Diode	D	
Crystal	X	
Inductor	L	
Power	VCC	
Ground	GND	

Example


Net label
Pin number
D[0..15]
Bus net
IC12
Reference designator
Pin name
A[1..20]
Off-page connector
Net
Junction
CPU_RESET
R39
33R
TP11
RESET
RX1
RX-1
TX1
TX-1
USB_INT
GPIO_1
GPIO_2
GPIO_3
P1.0/DCD0
P1.1/RTS0
/RTS1
VCC
D3
R39
33R
No connection between two nets
PXA255
Component type
GND
No connect
Net label showing connections
PL1
A2
A3
OUTPUT PORT

Memory Mapping

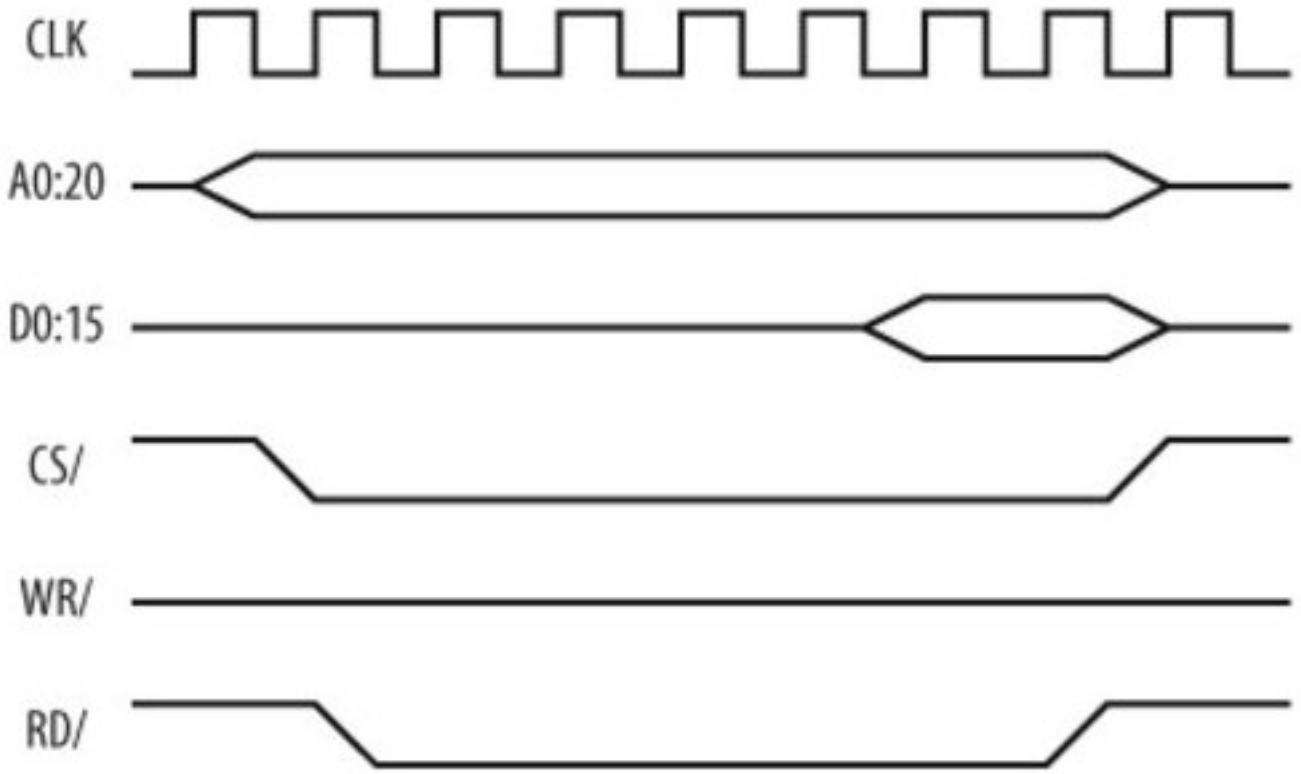
- How processor sees devices?
  - Memories
  - Peripherals
- Some boards have both memory and IO addressing spaces
- Some let devices to be mapped into memory
- Memory mapping often used for embedded devices

Memory Map

- Processor communicates with memory using
  - Address bus
  - Data bus
- Memory decoder decides for addresses which chip (Memory or IO) is used
- Control lines

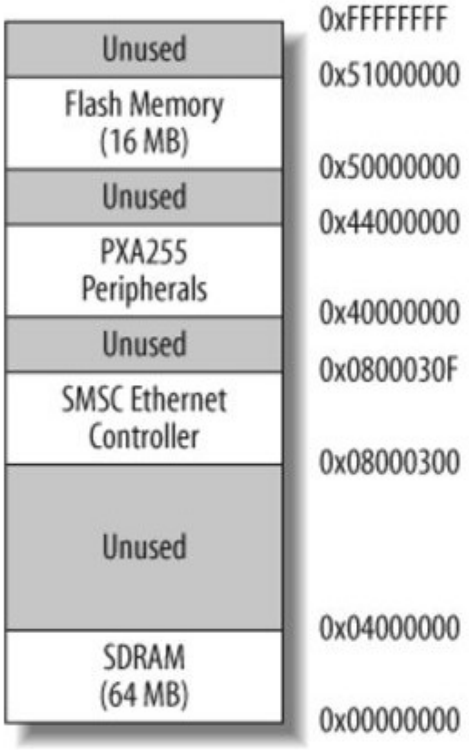
Timing diagram

- Timing diagram shows the relative timing of line activity in relation to the clock (CLK) signal.



Memory addressing

- When getting to know a system, note the address ranges of peripherals
- Create a table, highest at the top; C-header




```
/*    Base Address   Size  Description
*        0x00000000  64M  SDRAM
*        0x08000300  N/A  Ethernet controller
*        0x50000000  16M  Flash
*/
#define SDRAM_BASE (0x00000000)
#define ETHERNET_BASE (0x08000300)
#define FLASH_BASE (0x50000000)
```

How to Communicate?

- When addresses and names known, investigate communication
- Polling
- Interrupts
- Interrupts have overhead, but give cleaner code

Getting to Know the Processor

- Refer to databook
  - What address does the processor jump to after reset? What are the registers after reset?
  - In what sequence should peripheral registers be programmed
  - Where is the Interrupt Vector Table, how is it found?
  - What is its format?
  - Are there *traps*? Must there be an ISR for them
  - How are interrupts disabled
  - How are interrupts cleared

Example processor

- PXA255: CPU, interrupt control unit, memory controller, IO pins, four timers, 4 serial ports, 16 DMA channels, two pulse width modulators, real-time clock, power management, etc.
- Put the integrated peripheral addresses into header file:

```
/* Interrupt Controller Registers */
#define INTERRUPT_PENDING_REG (*((uint32_t volatile *)0x40D00000))
#define INTERRUPT_ENABLE_REG (*((uint32_t volatile *)0x40D00004))
#define INTERRUPT_TYPE_REG (*((uint32_t volatile *)0x40D00008))
```

Peripherals

- Study external hardware (datasheets again)
- How to communicate, the sequence, commands?
- Try to make the interface more abstract (readable)
  - Eventually write drivers for them

Initialization

- Written in assembly
- First thing processor does after reset, looks on some specific address. HW initialization jump there
- Done in stages

Stage 1

- hw_init routine
- Initialization of memory interface configuration registers: which memory devices are present

Stage 2

- startup (still in assembly)
- Set up stack for the system.
- Call main()